# CWE Detail – CWE-427

## Description

The product uses a fixed or controlled search path to find resources, but one or more locations in that path can be under the control of unintended actors.

## Extended Description

Although this weakness can occur with any type of resource, it is frequently introduced when a product uses a directory search path to find executables or code libraries, but the path contains a directory that can be modified by an attacker, such as "/tmp" or the current working directory. In Windows-based systems, when the LoadLibrary or LoadLibraryEx function is called with a DLL name that does not contain a fully qualified path, the function follows a search order that includes two path elements that might be uncontrolled: the directory from which the program has been loaded the current working directory In some cases, the attack can be conducted remotely, such as when SMB or WebDAV network shares are used. One or more locations in that path could include the Windows drive root or its subdirectories. This often exists in Linux-based code assuming the controlled nature of the root directory (/) or its subdirectories (/etc, etc), or a code that recursively accesses the parent directory. In Windows, the drive root and some of its subdirectories have weak permissions by default, which makes them uncontrolled. In some Unix-based systems, a PATH might be created that contains an empty element, e.g. by splicing an empty variable into the PATH. This empty element can be interpreted as equivalent to the current working directory, which might be an untrusted search element. In software package management frameworks (e.g., npm, RubyGems, or PyPi), the framework may identify dependencies on third-party libraries or other packages, then consult a repository that contains the desired package. The framework may search a public repository before a private repository. This could be exploited by attackers by placing a malicious package in the public repository that has the same name as a package from the private repository. The search path might not be directly under control of the developer relying on the framework, but this search order effectively contains an untrusted element.

## Threat-Mapped Scoring

Score: 2.65

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** CVE-2023-25815: chain: a change in an underlying package causes the gettext function to use implicit initialization with a hard-coded path (CWE-1419) under the user-writable C:\ drive, introducing an untrusted search path element (CWE-427) that enables spoofing of messages.

**•** CVE-2022-4826: Go-based git extension on Windows can search for and execute a malicious "..exe" in a repository because Go searches the current working directory if git.exe is not found in the PATH

**•** CVE-2020-26284: A Static Site Generator built in Go, when running on Windows, searches the current working directory for a command, possibly allowing code execution using a malicious .exe or .bat file with the name being searched

**•** CVE-2022-24765: Windows-based fork of git creates a ".git" folder in the C: drive, allowing local attackers to create a .git folder with a malicious config file

**•** CVE-2019-1552: SSL package searches under "C:/usr/local" for configuration files and other critical data, but C:/usr/local might be world-writable.

**•** CVE-2010-3402: "DLL hijacking" issue in document editor.

**•** CVE-2010-3397: "DLL hijacking" issue in encryption software.

**•** CVE-2010-3138: "DLL hijacking" issue in library used by multiple media players.

**•** CVE-2010-3152: "DLL hijacking" issue in illustration program.

**•** CVE-2010-3147: "DLL hijacking" issue in address book.

**•** CVE-2010-3135: "DLL hijacking" issue in network monitoring software.

**•** CVE-2010-3131: "DLL hijacking" issue in web browser.

**•** CVE-2010-1795: "DLL hijacking" issue in music player/organizer.

**•** CVE-2002-1576: Product uses the current working directory to find and execute a program, which allows local users to gain privileges by creating a symlink that points to a malicious version of the program.

**•** CVE-1999-1461: Product trusts the PATH environmental variable to find and execute a program, which allows local users to obtain root access by modifying the PATH to point to a malicous version of that program.

**•** CVE-1999-1318: Software uses a search path that includes the current working directory (.), which allows local users to gain privileges via malicious programs.

**•** CVE-2003-0579: Admin software trusts the user-supplied -uv.install command line option to find and execute the uv.install program, which allows local users to gain privileges by providing a pathname that is under control of the user.

**•** CVE-2000-0854: When a document is opened, the directory of that document is first used to locate DLLs , which could allow an attacker to execute arbitrary commands by inserting malicious DLLs into the same directory as the document.

**•** CVE-2001-0943: Database trusts the PATH environment variable to find and execute programs, which allows local users to modify the PATH to point to malicious programs.

**•** CVE-2001-0942: Database uses an environment variable to find and execute a program, which allows local users to execute arbitrary programs by changing the environment variable.

**•** CVE-2001-0507: Server uses relative paths to find system files that will run in-process, which allows local users to gain privileges via a malicious file.

**•** CVE-2002-2017: Product allows local users to execute arbitrary code by setting an environment variable to reference a malicious program.

**•** CVE-1999-0690: Product includes the current directory in root's PATH variable.

**•** CVE-2001-0912: Error during packaging causes product to include a hard-coded, non-standard directory in search path.

**•** CVE-2001-0289: Product searches current working directory for configuration file.

**•** CVE-2005-1705: Product searches current working directory for configuration file.

**•** CVE-2005-1307: Product executable other program from current working directory.

**•** CVE-2002-2040: Untrusted path.

**•** CVE-2005-2072: Modification of trusted environment variable leads to untrusted path vulnerability.

**•** CVE-2005-1632: Product searches /tmp for modules before other paths.

## Related Attack Patterns (CAPEC)

* CAPEC-38
* CAPEC-471

## Attack TTPs

**•** T1574.007: Path Interception by PATH Environment Variable (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.001: DLL (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.008: Path Interception by Search Order Hijacking (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.009: Path Interception by Unquoted Path (Tactics: persistence, privilege-escalation, defense-evasion)

**•** T1574.004: Dylib Hijacking (Tactics: persistence, privilege-escalation, defense-evasion)

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands — Notes:

## Potential Mitigations

**•** Architecture and Design: Hard-code the search path to a set of known-safe values (such as system directories), or only allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-426 and CWE-428. (Effectiveness: N/A)

**•** Implementation: When invoking other programs, specify those programs using fully-qualified pathnames. While this is an effective approach, code that uses fully-qualified pathnames might not be portable to other systems that do not use the same pathnames. The portability can be improved by locating the full-qualified paths in a centralized, easily-modifiable location within the source code, and having the code refer to these paths. (Effectiveness: N/A)

**•** Implementation: Remove or restrict all environment settings before invoking other programs. This includes the PATH environment variable, LD\_LIBRARY\_PATH, and other settings that identify the location of code libraries, and any application-specific search paths. (Effectiveness: N/A)

**•** Implementation: Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory. Since this is a denylist approach, it might not be a complete solution. (Effectiveness: N/A)

**•** Implementation: Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of finding the program using the PATH environment variable, while execl() and execv() require a full path. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** The problem here is that the program does not specify an absolute path for make and does not clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

**•** Therefore, Go would prioritize searching the current directory when the provided command name does not contain a directory separator and continued to search for programs even when the specified program name is empty.

**•** The researcher discovered the names of some internal, private packages by looking at dependency lists in public source code, such as package.json. The researcher then created new, untrusted packages with the same name as the internal packages, then uploaded them to package hosting services. These services included the npm registry for Node, PyPi for Python, and RubyGems. In affected companies, their dependency resolution would search the public hosting services first before consulting their internal service, causing the untrusted packages to be automatically downloaded and executed.

## Notes

**•** Relationship: Unlike untrusted search path (CWE-426), which inherently involves control over the definition of a control sphere (i.e., modification of a search path), this entry concerns a fixed control sphere in which some part of the sphere may be under attacker control (i.e., the search path cannot be modified by an attacker, but one element of the path can be under attacker control).

**•** Theoretical: This weakness is not a clean fit under CWE-668 or CWE-610, which suggests that the control sphere model might need enhancement or clarification.